Graphische Datenverarbeitung II

Mathematische Grundlagen & Visualisierungstechniken I

Prof. Dr. Elke Hergenröther

Mathematische Grundlagen: Transformationen

Punkt:

Punkte in der Ebene werden durch ihre x- und y-Koordinaten festgelegt. Im weiteren werden Punkte als Spaltenvektoren geschrieben:

$$\vec{P} = \begin{bmatrix} x_P \\ y_P \end{bmatrix}$$

Mathematische Grundlagen: Transformationen

Gerade, Strahl und Strecke:

Gerade, Strahl und Strecke werden in der parametrischen Form eingeführt, da man hiermit jeden Punkt gezielt ansteuern kann.

Die Parameterform
- erlaubt die effiziente Berechnung von Schnittpunkten,
- wird für Raytracing eingesetzt,
- usw.

Mathematische Grundlagen: Transformationen

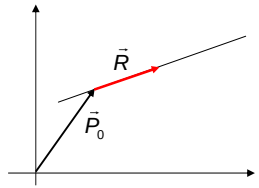
Die (gerichtete) Strecke:

Die gerichtete Strecke zwischen zwei Punkten $\vec{P}_0$ und $\vec{P}_1$ ist die Menge aller Punkte $\vec{x}$, für die gilt:

$$\vec{X}(u) = \vec{P}_0 + u \cdot (\vec{P}_1 - \vec{P}_0) = \vec{P}_0 + u \cdot \vec{R}$$

mit $u \in [0,1]$ und $\vec{R}$ ist Richtungsvektor

von $\vec{P}_0$ nach $\vec{P}_1$



Mathematische Grundlagen: Transformationen

Strecke, Strahl und Gerade:

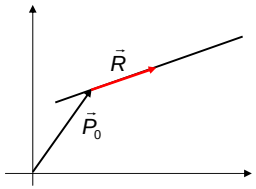
wenn für die Parameterform $\vec{X}(u) = \vec{P}_0 + u \cdot (\vec{P}_1 - \vec{P}_0) = \vec{P}_0 + u \cdot \vec{R}$

mit $\vec{R}$ ist Richtungsvektor von $\vec{P}_0$ nach $\vec{P}_1$ gilt:

$u \in [0,1]$ dann beschriebt sie eine Strecke

$u \geq 0$ dann beschreibt sie einen Strahl

$u \in IR$ dann beschreibt sie eine Gerade



Mathematische Grundlagen: Transformationen

Gerade, Strahl und Strecke:

Die Parameterform in Matrixschreibweise:

$$\vec{X}(u) = \vec{P}_0 + u \cdot (\vec{P}_1 - \vec{P}_0) = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + u \cdot \begin{bmatrix} x_1 - x_0 \\ y_1 - y_0 \end{bmatrix} = \begin{bmatrix} x_0 + ux_1 - ux_0 \\ y_0 + uy_1 - uy_0 \end{bmatrix} = \begin{bmatrix} x_0(1-u) + ux_1 \\ y_0(1-u) + uy_1 \end{bmatrix}$$

$$\vec{X}(u) = \begin{bmatrix} x_0(1-u) + ux_1 \\ y_0(1-u) + uy_1 \end{bmatrix} = \begin{bmatrix} x_0 & x_1 \\ y_0 & y_1 \end{bmatrix} \cdot \begin{bmatrix} 1-u \\ u \end{bmatrix} = \begin{bmatrix} x(u) \\ y(u) \end{bmatrix}$$

Koordinatentransformationen in 2D

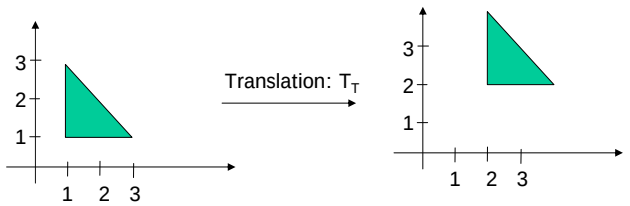
- Translation (Verschiebung)
- Skalierung (Größenanderung)
- Rotation (Drehung)

Translation (Verschiebung)

Translation eines Punktes $\vec{P} = [x_P, y_P]^t$ um dx in x- und um dy in y-Richtung:

$x_P` = dx + x_P$
$y_P` = dy + y_P$

d.h. $\begin{bmatrix} x_P' \\ y_P' \end{bmatrix} = \begin{bmatrix} dx \\ dy \end{bmatrix} + \begin{bmatrix} x_P \\ y_P \end{bmatrix}$ bzw. $\vec{P}' = T_T + \vec{P}$

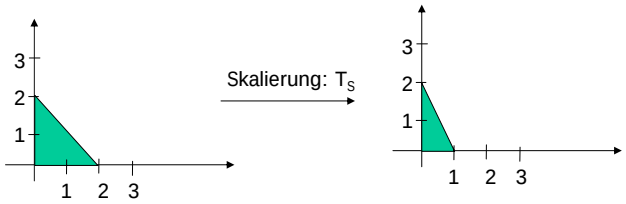


Skalierung (Größenänderung)

Skalierung eines Punktes $\vec{P} = [x_P, y_P]^t$ mit s_x in x- und um s_y in y-Richtung:

$x_P` = s_x * x_P$
$y_P` = s_y * y_P$

d.h. $\begin{bmatrix} x_P' \\ y_P' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x_P \\ y_P \end{bmatrix}$ bzw. $\vec{P}' = T_S + \vec{P}$



Rotation (Drehung)

Rotation eines Punktes $\vec{P} = [x_P, y_P]^t$ um den Winkel α bezüglich des Ursprungs:

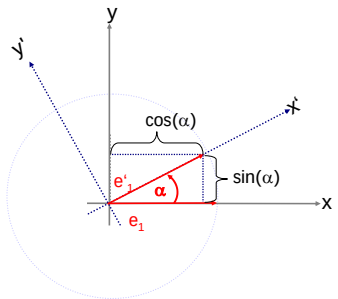
$x_P` = \cos(\alpha)*x_P - \sin(\alpha)*y_P$
$y_P` = \sin(\alpha)*x_P + \cos(\alpha)*y_P$

d.h. $\begin{bmatrix} x_P' \\ y_P' \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix} \cdot \begin{bmatrix} x_P \\ y_P \end{bmatrix}$ bzw. $\vec{P}' = T_R + \vec{P}$

Beispiel: Rotation um 90^0: $\begin{bmatrix} x_P' \\ y_P' \end{bmatrix} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 4 \end{bmatrix} = \begin{bmatrix} -4 \\ 2 \end{bmatrix}$

Wie kommt man auf die Rotationsmatrix?

Nicht das Objekt wird rotiert, sondern das Koordinatensystem:



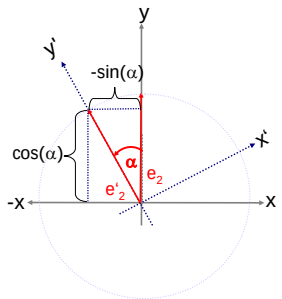
Bei einer Rotation um den Winkel α um die z-Achse in positiver Richtung $R_z(\alpha)$ werden die Einheitsvektoren des kartesischen Koordinatensystems auf die folgenden Basisvektoren des affinen Koordinatensystems abgebildet:

$$R_z(\alpha) \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \\ 0 \end{pmatrix}$$

z = z', wird daher nicht eingezeichnet

Affine Transformationen: Rotation um die Z-Achse

Winkel im Einheitskreis:



$$R_z(\alpha) \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \\ 0 \end{pmatrix}$$

$$R_z(\alpha) \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -\sin(\alpha) \\ \cos(\alpha) \\ 0 \end{pmatrix}$$

$$R_z(\alpha) \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

$$R_z(\alpha) = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Homogene Koordinaten

Da es sinnvoll ist die verschiedenen Transformationen miteinander zu einer akkumulierten Matrix zu „verrechnen“und dann auf alle Eckpunkte anzuwenden (siehe OpenGL), muss es eine Möglichkeit geben, den Additivanteil (Translation) und den Multiplikativanteil (Rotation, Skalierung) miteinander zu verknüpfen:

Lösung: Homogene Koordinaten

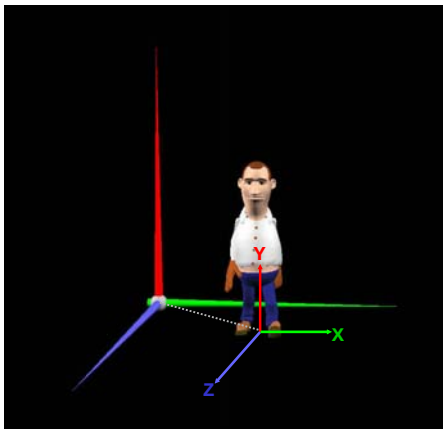
Homogene Koordinaten

Die Vektoren $\vec{x}'$, $\vec{y}'$und $\vec{t}$ werden zu einer Matrix T (Transformationsmatrix) zusammengefasst:

$$\begin{pmatrix} p'_1 \\ p'_2 \end{pmatrix} = \begin{pmatrix} x_1' & y_1' \\ x_2' & y_2' \end{pmatrix} \cdot \begin{pmatrix} p_1 \\ p_2 \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \end{pmatrix} \Leftrightarrow \begin{pmatrix} p'_1 \\ p'_2 \\ 1 \end{pmatrix} = \begin{pmatrix} x_1' & y_1' & t_1 \\ x_2' & y_2' & t_2 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} p_1 \\ p_2 \\ 1 \end{pmatrix}$$

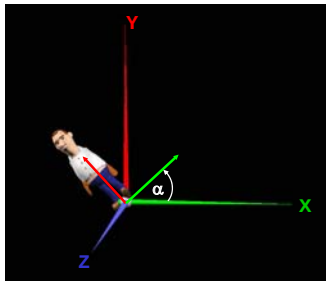
Homogene Koordinaten

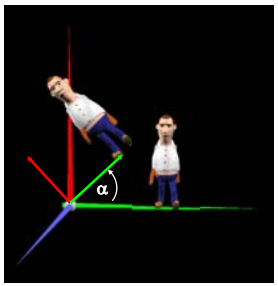
Transformationen: Translation



$$\begin{pmatrix} p'_1 \\ p'_2 \\ p'_3 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_1 \\ 0 & 1 & 0 & t_2 \\ 0 & 0 & 1 & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ 1 \end{pmatrix}$$

Transformationen: Rotation um die Z-Achse





$$R_z(\alpha) = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Transformation: Rotation um die x- und y-Achse

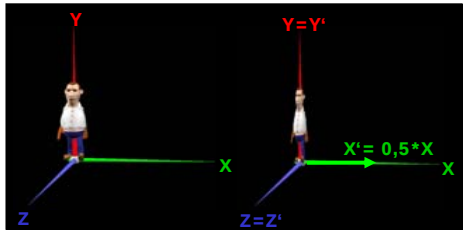
Rotation um die x-Achse:

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation um die y-Achse:

$$R_y(\alpha) = \begin{pmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

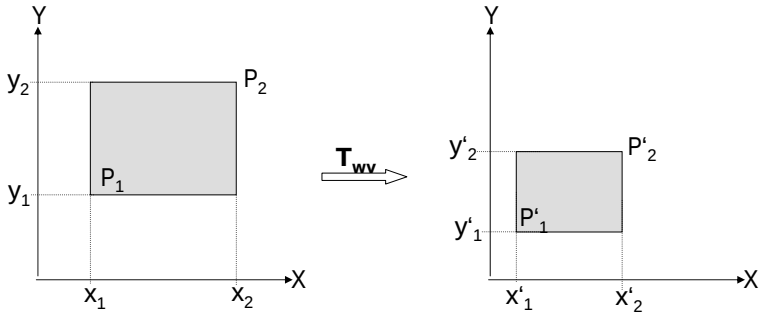
Affine Transformation: Skalierung



$$S\begin{pmatrix} s_x \\ s_y \\ s_z \end{pmatrix} = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Beispiel einer zusammengesetzten Transformation:

Window-Viewport-Transformation



Visualisierungstechniken

Visualisierung:

Visualisierung bedeutet sichtbar machen, darstellen. Die CG beschränkt sich dabei jedoch nicht auf die Abbildung real existierender sondern beschäftigt sich auch mit der Darstellung von Information und wissenschaftlich-technischen Daten.

Visualisierung geometrischer Modelle

- Visualisierung ohne Berücksichtigung von Licht
- Visualisierung mit Berücksichtigung von Licht
  - lokale Verfahren
  - globale Verfahren

Visualisierung ohne Berücksichtigung von Licht

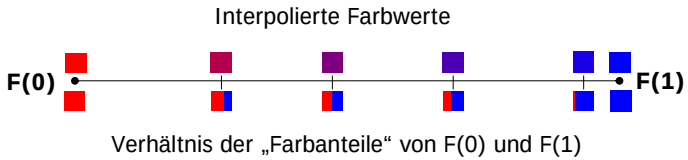
- Darstellen eines Primitivs durch setzen eines Farbattributs:
  - rote Fläche,
  - grüne Linie
- Was passiert, wenn jedem Eckpunkt (Vertex) eine andere Farbe zugeordnet wird?

Visualisierung ohne Berücksichtigung von Licht

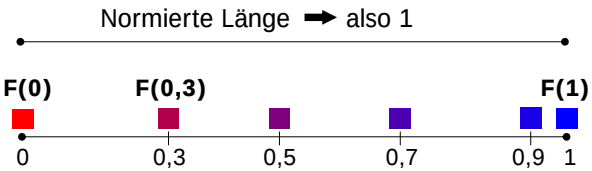
Lineare Interpolation:
Interpolationsverfahren für Linien

Bilineare Interpolation:
Interpolationsverfahren für Flächen

Lineare Farbinterpolation



Lineare Farbinterpolation



Beispiel: **F(0,3)** = 0,7 * **F(0)** + 0,3 * **F (1)**

70% Rot 30% Blau

Allgemeine Formel: **F(t)** = (1-t) * **F(0)** + t * **F (1)**

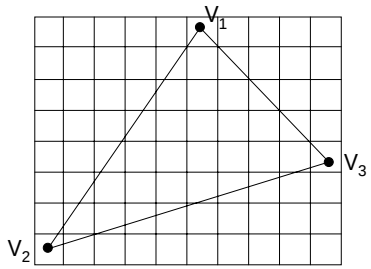
Lineare Farbinterpolation

$$F(t) = (1-t)\cdot F(0) + t\cdot F(1)$$

$$F(t) = (1-t)\cdot \begin{pmatrix} R(0) \\ G(0) \\ B(0) \end{pmatrix} + t\cdot \begin{pmatrix} R(1) \\ G(1) \\ B(1) \end{pmatrix}$$

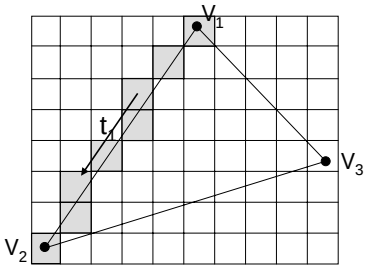
Bilineare Farbinterpolation für Dreiecke

1. Schritt: Rasterisierung des Dreiecks



Bilineare Farbinterpolation für Dreiecke

2. Schritt: Farbwerte der Pixel zwischen den Vertices interpolieren

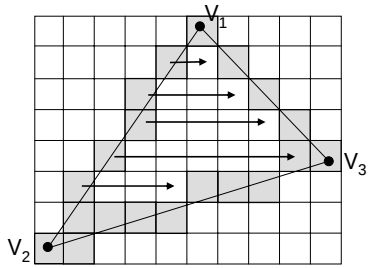


Interpolation zwischen V_1 und V_2:

- Zunächst muss die Strecke zwischen den beiden Vertices normiert werden.
- Der Parameter t_1 steht für die normierte Strecke und geht von 0 bis 1.
- Danach erfolgt eine lineare Farbinterpolation abhängig von t_1.

Bilineare Farbinterpolation für Dreiecke

Farbwerte der Pixel zwischen den Vertices wurden linear interpoliert



3. Schritt:

Lineare Interpolation der Pixelwerte entlang der einzelnen Rasterzeilen

Ergebnis einer bilinearen Farbinterpolation